Applicant thanks the Examiner for the thorough examination of the subject application. In the subject election/restriction requirement, the Examiner has required the restriction of the application to one of the following groups of claims:

Group I:	Claims 4-6, 8-14, 20, 22, 24-27
Group II:	Claims 15-18

Under the provisions of 37 C.F.R. 1.143, Applicant herein provisionally elects, with traverse, the invention of Group I, including claims 4-6, 8-14, 20, 22, 24-27 for prosecution. Consistent with this provisional election, Applicant has not, as of yet, cancelled the non-elected claims (namely claims 15-18).

Applicant reminds the examiner that a restriction requirement may be proper under 35 U.S.C. §121 if two or more "independent and distinct" inventions are claimed in one application. Additional support for this may be found in 37 C.F.R. §1.141, which states that two or more "independent and distinct inventions" may not be claimed in one application.

In this case, the Examiner appears to be attributing meaning to a statement made by Applicant in the response dated October 27, 2008, which is far outside the statement's intended purpose. By way of background, on page 5 of the Official Action dated 7/25/2008, the Examiner rejected Applicant's claims 15 and 16 under 35 U.S.C. §103 by merely stating that these claims were "substantially the same" as Applicant's claim 4. The Examiner provided no other support for this rejection. Claims 4 and 15-16 are provided below for the Examiner's convenience.

4. (Previously Presented) A method of verifying a program fragment downloaded onto a reprogrammable embedded system, equipped with a rewritable memory, a microprocessor and a virtual machine equipped

with an execution stack and with operand registers, said program fragment consisting of an object code and including at least one subprogram consisting of a series of instructions manipulating said operand registers, said microprocessor and virtual machine configured to interpret said object code, said embedded system being interconnected to a reader, wherein subsequent to a detection of a downloading command and a storage of said object code in said rewritable memory, said method, for each subprogram, comprises:

initializing a type stack and a table of register types through data representing a state of said virtual machine on initialization of an execution of said temporarily stored object code;

carrying out a verification process of said temporarily stored object code instruction by instruction, by discerning an existence, for each current instruction, of a target, a branching-instruction target, a target of an exception-handler call or a target of a subroutine call, and, said current instruction being a target of a branching instruction, said verification process including verifying that said stack is empty and rejecting said program fragment otherwise;

verifying and updating an effect of said current instruction on said data types of said type stack and of said table of register types;

said verification process being successful when said table of register types is not modified in the course of a verification of all said instructions, and said verification process being carried out instruction by instruction until said table of register types is stable, with no modification being present, said verification process being interrupted and said program fragment being rejected, otherwise.

15. (Currently Amended) A method of transforming an object code of a program fragment including a series of instructions, in which operands of each instruction belong to data types manipulated by said instruction, an execution stack does not exhibit any overflow phenomenon, and for each branching instruction, a type of stack variables at a corresponding branching is the same as that of targets of branching, into a standardized object code for this same program fragment, wherein, for all said instructions of said object code, said method comprising:

annotating each current instruction with a data type of said stack before and after execution of said current instruction, with said annotation data being calculated by means of an analysis of a data stream relating to said current instruction;

detecting, within said instructions and within each current instruction, an existence of branchings, or of branching-targets, for which said execution stack is not empty, said detecting operation being carried out on a basis of the annotation data of said type of stack variables allocated to each current instruction; and in case of detection of a non-empty execution stack,

inserting instructions to transfer stack variables on either side of said branchings or of said branching targets respectively, in order to empty contents of said execution stack into temporary registers before said branching and to reestablish said execution stack from said temporary registers after said branching; and not inserting any transfer instruction otherwise, said method allowing to obtain a standardized object code for said same program fragment, in which said operands of each instruction belong to said data types manipulated by said instruction, said execution stack does not exhibit any overflow phenomenon, said execution stack is empty at each branching instruction and at each branching—target instruction, in an absence of any modification to the execution of said program fragment.

16. (Currently Amended) A method of transforming an object code of a program fragment including a series of instructions, in which operands of each instruction belong to data types manipulated by said instruction, and at least one of the operands of a given type are written into a register by an instruction of object code ~~is~~ reread from said register by another instruction of said object code with a same given data type, into a standardized object code for the program fragment, wherein for all said instructions of said object code, said method comprising:

annotating each current instruction with said data type of said register~~s~~ before and after execution of said current instruction, with said annotation data being calculated by means of an analysis of a data stream relating to said instruction;

carrying out a reallocation of said registers, by detecting at least one of original registers employed with different types, dividing these original registers into separate standardized registers, with one standardized register for each data type used, and

reupdating said instructions which manipulate said operands which use said standardized registers;

said method configured to obtain said standardized object code for said program fragment in which said operands of each instruction belong to said data types manipulated by said instruction, said data type being allocated to a same register throughout said standardized object code.

Applicant notes that independent claim 4 includes the affirmative limitations of "initializing", "carrying", and "verifying", claim 15 includes the affirmative limitations of "annotating", "detecting", and "inserting", and claim 16 includes the affirmative limitations of "annotating", "carrying", and "reupdating". In addressing the Examiner's rejection, Applicant stated in the response dated October 27, 2008 "Applicant disagrees that the claims are 'substantially the same' as indicated by the Examiner".

As shown above, independent claims 4, 15, and 16 include different limitations and, as such, it is Applicant's understanding that they are not "substantially the same." It is Applicant's belief that this rejection under 35 U.S.C. §103 was improper as the claims clearly were not "substantially the same" and, as such, deserved a more thorough examination.

Applicant respectfully submits that method claims 15-17 may be used to guarantee that software modified by these methods will pass the validation steps of the claims of Group I when downloaded as it complies to test conditions such as "no stack overflow", "execution stack empty", etc. Support for this proposition may be found throughout the subject application, for example, on page 7, portions of which have been provided below for the Examiner's convenience.

> Another subject of the present invention is also the implementation of a method of verifying a downloaded program fragment, or applet, in which a process of static verification of the code of the applet is conducted when it is downloaded, this process possibly being similar, at least in its principle, to the third solution described above, but into which new verification techniques are introduced, so as to allow execution of this verification within the memory size and computation speed value limits imposed by the microprocessor cards and other low-power embedded data-processing systems.
>
> Another subject of the present invention is also the implementation of methods of transforming program fragments of conventional type obtained,

> for example, by the use of a JAVA compiler, into standardized program fragments, or applets, satisfying, a priori, the verification criteria of the verification method which is the subject of the invention, with a view to accelerating the process of verifying and executing the latter in present-day microprocessor cards or embedded data-processing systems. *Subject application, p. 7, lines 1-22.*

Further, Applicant reminds the Examiner that the correct standard for issuing a restriction requirement under 35 U.S.C. §121 is if two or more "independent and distinct" inventions are claimed in one application, not whether or not the claims at issue are "substantially the same". Moreover, the Examiner states "Applicant admits the claims are not related in the remarks of 10/27/2008." *Official Action dated 1/6/09, page 2.* Applicant respectfully disagrees with this characterization of Applicant's statements. Applicant did not admit that the claims "are not related" or that the claims are independent and distinct. In contrast, Applicant merely stated that claims 4, 15, and 16 were not "substantially the same" for purposes of the 35 U.S.C. § 103 rejection and, as such, required a more thorough examination.

In light of the above, Applicant respectfully requests that the restriction requirement under 35 U.S.C. §121 be withdrawn. Applicant believes there are no fees due at this time. Please apply any charges or credits to Deposit Account No. 50-2324, referencing attorney docket no. 102114.00034.

Respectfully submitted,

Date: 06 March 2009

/Brian J Colandreo/
Brian J Colandreo
Reg. No. 42,427

Holland & Knight LLP
10 St. James Avenue
Boston, MA 02116
Tel (617) 305-2143
Fax (617) 523-6850

6140729_v1